



## Contents

- [1 Description](#)
- [2 Sample layouts](#)
- [3 Implementation](#)
  - ◆ [3.1 The layout HTML structure](#)
  - ◆ [3.2 The basic CSS code](#)
  - ◆ [3.3 The JavaScript code](#)
    - ◇ [3.3.1 The layout properties](#)
    - ◇ [3.3.2 Buttons/SoftKeys generation](#)
    - ◇ [3.3.3 Laying out the layout elements](#)
    - ◇ [3.3.4 Managing display rotations](#)
    - ◇ [3.3.5 Setting the widget main view](#)
    - ◇ [3.3.6 Initializing layout and events](#)
    - ◇ [3.3.7 Polling for display rotations](#)
- [4 Layout object: how to use it](#)
- [5 Downloads](#)

## Description

This article explains **how to build from scratch a layout for Web Runtime widgets** with these **UI features**:

- a **header bar**, that can contain the widget name and other optional elements
- a **central body**, containing the dynamic content of the widget
- a **buttons' toolbar**

Also, the built layout will support the following **functionalities**:

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

- **unified method to add buttons or softkeys**, depending on the widget needs (e.g.: based on device's display resolution)
- **support for display rotations**
- **repositioning of buttons' toolbar** depending on display orientation
- support for **easy switching** of the widget main view
- automatic **support for scrolling**

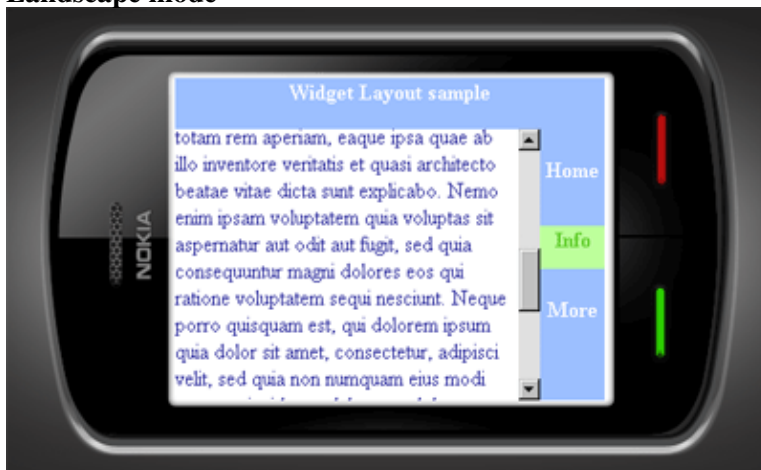
## Sample layouts

The following pictures shows **how the layout appears in portrait and landscape mode**:

### Portrait mode



### Landscape mode



## Implementation

### The layout HTML structure

The layout is structured into these **main DOM elements**:

- a **DIV element for the header**
- a **DIV element for the widget content**
- **2 DIV elements for the buttons' toolbar**, that will be used depending on the display orientation

The basic layout structure is the following:

```
<html>
<body>

<div id="header">

</div>

<div id="right_toolbar">

</div>

<div id="screen_container">

</div>

<div id="bottom_toolbar">
<div id="toolbar_buttons">
</div>
</div>

</body>
</html>
```

The DIV element with **toolbar\_buttons** id is a placeholder element for the toolbars' buttons.

### The basic CSS code

In order to properly layout the various interface elements, very **few basic CSS rules** are needed:

```
#right_toolbar
{
float:right;
}
#screen_container
{
overflow-y
```

Things to note in the above CSS code:

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

- the **right\_toolbar** is set to **float: right**, in order to **have the toolbar, on landscape displays, on the right side of the widget**. In order to have it on the left side, it is enough to change the float property to left.

An **alternative approach** to have the buttons' toolbar differently placed depending on the display orientation, **based on absolute positioning**, is visible in the **STEW widget**, and is explained in details here: [STEW: supporting screen rotation](#)

- the **overflow-y: auto** property for the **screen\_container** allow to support **vertical scrolling for views** that are higher than the available space.

## The JavaScript code

All the widget layout is managed through a **layout JavaScript object**, that will be defined and implemented in this section. This object takes care of **layout appearance, events management, and view switching**.

### The layout properties

First, some **properties** are set in order to control the **layout behavior and appearance**.

```
var layout =
{
/* true if the layout must use softkeys
   * false if the layout must use toolbar buttons
   */
  useSoftkeys:

/* CSS class applied to the buttons' toolbar */
  toolbarButtonCssClass: 'toolbar_button',

/* value of the last detected display width */
  lastDetectedWidth:

/* value of the last detected display height */
  lastDetectedHeight:

/* holds a reference to the currently displayed view */
  currentScreen:
}
```

### Buttons/SoftKeys generation

Depending on the **useSoftkeys** property value, the layout will **generate and handle softkey menu items or toolbar buttons**, that will be inserted in the **toolbar\_buttons** placeholder DIV element. In order to do this, the following **addMainOption()** method is defined:

```
var layout =
{
[...]
```

```
    addMainOption(text, id, handler)
{
```

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

```
if(this.useSoftkeys)
{
var item = new MenuItem(text, id);

item.onSelect = handler;

    append(item);        menu.
}
else
{
var button = document.createElement('div');

    className = this.toolbarButtonCssClass;

    appendChild(document.createTextNode(text));

    onclick = handler;button.

    getElementById(document.getElementById('toolbar_buttons')).appendChild(button);
}
},
}
```

The three arguments passed to the **addMainOption()** method are:

- the **menu item/button label**,
- the **menu item id** (id is not used for toolbar buttons)
- the **handler function** called on the select/click events

### Laying out the layout elements

First, a helper methods is defined in order to detect display orientation:

```
var layout =
{
[...]
```

```
    isLandscape()
{
return window.innerWidth > window.innerHeight;
}
}
```

The layout object has to **manage the widget's appearance**, and specifically has to perform these operations:

- **resize the various layout elements** depending on the display size and orientation
- **place the toolbar buttons** (if not using softkeys) according to the display orientation

All these operations are managed by the following **setupInterface()** method:

```
var layout =
{
[...]
```

```
    setupInterface()
{
if(!this.useSoftkeys)
    hideSoftkeys();    menu.
}
}
```

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

```
this.lastDetectedWidth = window.innerWidth;
this.lastDetectedHeight = window.innerHeight;

if(this.useSoftkeys)
{
    getElementById('documentcontainer').style.height =
(this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';
}
else if(this.isLandscape())
{
    getElementById('documenttoolbar').style.display = 'none';
    getElementById('documenttoolbar').style.display = '';
    getElementById('documenttoolbar').appendChild(document.getElementById('toolbar_buttons'));

    getElementById('documenttoolbar').style.height =
(this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';

    getElementById('documentcontainer').style.height =
(this.lastDetectedHeight - document.getElementById('header').offsetHeight) + 'px';
}
else
{
    getElementById('documenttoolbar').style.display = '';
    getElementById('documenttoolbar').style.display = 'none';
    getElementById('documenttoolbar').appendChild(document.getElementById('toolbar_buttons'));

    getElementById('documentcontainer').style.height =
(this.lastDetectedHeight - document.getElementById('header').offsetHeight - document.getElementBy
```

### Managing display rotations

**When the display changes orientation**, the interface needs to be relayed out, in order to **resize and move elements appropriately**. Since all these things are performed by the `setupInterface()` method, the following `onResize()` method will uniquely call this method:

```
var layout =
{
    [...]

    onResize: function()
    {
        this.setupInterface();
    },
}
```

### Setting the widget main view

In order to **easily change the main view of the widget**, the `gotoScreen()` is defined, that will hide the currently visible view, and show the new one.

```
var layout =
{
    [...]
```

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

```
        gotoScreen(screenId, clearHistory)
    {
    var newScreen = document.getElementById(screenId);

    if(newScreen)
    {
    if(this.currentScreen != null)
    {
    this.currentScreen.style.display = 'none';
    }
    this.currentScreen = newScreen;

    this.currentScreen.style.display = '';
    }
    }
    }
```

### Initializing layout and events

Now, **all is ready to be initialized and used**. So, it is possible to define an **init()** method that will take care of:

- **calling methods for layout initialization**
- **setting up event handlers** for widget resize events

```
var layout =
{
[...]
```

```
    : function(startScreen)
    {
    this.setupInterface();

    var self = this;

        addEventListener(
        'resize',
        function()
        {
            onResize();                self.
        },
        false
    );

    this.gotoScreen(startScreen);
    }
    }
```

### Polling for display rotations

**Some devices**, as explained [here](#), **do not support the 'onResize' event**. So, the approach defined in the linked Forum Nokia Library page is implemented, in order to correctly manage display resize events also on these devices. The following **pollResize()** method is defined:

```
var layout =
```

Setting the widget main view

## How\_to\_build\_a\_Web\_Runtime\_layout\_with\_Header\_and\_Buttons\_Bar

```
{
[...]
```

```
    pollResize()
{
if(window.innerWidth != this.lastDetectedWidth || window.innerHeight != this.lastDetectedHeight)
{
this.onResize();
}
}
}
```

And is **scheduled to be periodically called** by the **init()** method, that is modified as follows:

```
var layout =
{
[...]
```

```
    : function(startScreen)
{
[...]
```

```
        (    setInterval
function()
{
    pollResize();        self.
}
'
        1000
);
}
}
```

## Layout object: how to use it

Usage of the **layout** object is quite straightforward, and requires these steps:

- **import the layout JavaScript and CSS files** (available here: [Media:WRTLLayoutHeaderAndButtons\\_JSandCSS.zip](#))

) in your widget's HTML code:

```
<html>
<head>
<script language="javascript" type="text/javascript" src="layout.js"></script>
<link rel="stylesheet" href="layout.css" type="text/css">
</head>

[...]
```

- **define the basic HTML structure**, as defined in the above section, and complete them with your widget specific code. An example of widget HTML code with 3 main views is the following:

```
<html>
[...]
```

## How to build a Web Runtime layout with Header and Buttons Bar

```
<body onLoad="javascript:init();">

<div id="header">
    Widget Layout sample
</div>

<div id="right_toolbar">
</div>

<div id="screen_container">

<div class="screen" id="screen_home" style="display: none;">
<strong>Widget Layout with Header and Buttons' Toolbar</strong>
</div>

<div class="screen" id="screen_info" style="display: none;">
<h2>Widget Info</h2>

    Some infos about the application

</div>

<div class="screen" id="screen_more" style="display: none;">
    Some more info...
</div>
</div>

<div id="bottom_toolbar">
<div id="toolbar_buttons">

</div>
</div>

</body>
</html>
```

- then, the **layout can be initialized**, by **adding the desired softkeys/buttons** and by calling its initialization method.

```
function init()
{
    useSoftkeys = false;

    addMainOption('Home', 1001, function(){layout.gotoScreen("screen_home");})
    addMainOption('Info', 1002, function(){layout.gotoScreen("screen_info");})
    addMainOption('More', 1003, function(){layout.gotoScreen("screen_more");})

    layout.gotoScreen('Home');
}
```

So this example layout does not use softkeys: 3 buttons are so created, and each of them brings to a view defined in the above HTML code. Screenshots of the sample widget are visible at the beginning of this article.

## Downloads

The following related files are available for download:

- A sample widget using the layout implemented in this article:  
[Media:WRTLAYOUTHeaderAndButtons.zip](#)
- The layout JavaScript and CSS files: [Media:WRTLAYOUTHeaderAndButtons\\_JSandCSS.zip](#)